Aulas e Provas de Avaliação

Estruturas de Dados e algoritmos **Formulário**

O meu primeiro programa em C++:

```
#include <iostream>   // para poder utilizar cout
using namespace std;  // para nao ter de escrever std::cout
int main() {
  cout << "Bom dia!" << endl;
  return(0);
}
```

Operadores aritméticos

Operador	Significado
*	multiplicação
/	divisão
+	adição
-	subtracção
%	resto da divisão inteira
++	incremento unitário
--	decremento unitário

Atribuição e operadores aritméticos

Operador	Significado
=	atribuição
*=	multiplicação e atribuição
/=	divisão e atribuição
+=	adição e atribuição
-=	subtracção e atribuição
%=	resto da divisão inteira e atribuição

Tabela 1: Operadores aritméticos e Atribuição.

Operadores relacionais

Operador	Significado
<=	menor ou igual que
<	menor que
>=	maior ou igual que
>	maior que
==	igual
!=	diferente

Operadores Lógicos

Operador	Significado	Utilização
||	OU lógico ($\vee$)	(exp1) || (exp2)
&&	E lógico ($\wedge$)	(exp1) && (exp2)
!	Negação ($\neg$)	! (exp1)

Tabela 2: Operadores relacionais e lógicos.

Operadores Lógicos Bit a Bit

Operador	Significado
&	E lógico bit a bit
|	OU lógico bit a bit
^	OU exclusivo bit a bit
~	Complemento bit a bit
<<	Deslocamento dos bits à esquerda
>>	Deslocamento dos bits à direita

Tabela 3: Operadores relacionais e lógicos bit a bit

Estes operadores são válidos sobre dados do tipo **int** ou **char**.

Tipo	tamanho em octetos (depende do equipamento)
bool	1
char	1
short int	2
int	4
long int	4
float	4
double	8
long double	12

Tabela 4: Tipos básicos.

`\b`	*backspace*	Mover o cursor uma vez para trás
`\f`	*form feed*	Ir para o início de uma nova página
`\n`	*new line*	Ir para a linha seguinte
`\r`	*return*	Ir para o início da linha corrente
`\t`	*tab*	Ir para a posição de tabulação seguinte
`\'`	*apóstrofe*	O símbolo '
`\"`	*aspas*	O símbolo "
`\\`	*barra à esquerda*	O símbolo \
`\a`	*bell (alarm)*	Emite um som (apito)
`\nnn`	*O carácter cujo código em octal é nnn*	Apresenta esse carácter
`\xNN`	*O carácter cujo código em hexadecimal é NN*	Apresenta esse carácter

Tabela 5: Caracteres especiais.

Forma geral de declaração e definição para uma variável de um tipo básico:

`<tipo>` *nome; // comentário*
`<tipo>` *nome = valor; // comentário*
`<tipo>` & *outronome = nome;* // variável do tipo referência
`<tipo>` * *nome_ptr;* // variável do tipo ponteiro
`<tipo>` * *nome_ptr* = `nullptr;` // sem apontar
`<tipo>` * *nomeX_ptr* = `&` *nomeX;* // variável do tipo ponteiro

em que `<tipo>` deve ser substituído por um dos existentes na tabela 5, e `valor` deverá ser uma constante, ou uma expressão adequada à inicialização da variável em causa; o tipo de *nomeX* deverá igual ao tipo de **nomeX_ptr*.

Declaração e definição de uma estrutura (definição de um novo tipo):

```
struct <nome_estrutura> {
   <tipo> membro1;
   <tipo> membro2;
   . . .
   <tipo> membroN;
};
```

Para aceder ou atribuir valor a um dos membros utiliza-se a sintaxe:
nome_var.membro // no caso da estrutura
nome_var_ptr`->`*membro* // no caso de ponteiro para estrutura
(nome_var_ptr).membro* // equivalente à anterior (não se usa)

O tipo enum:
`enum` *nome_enum* `{`*etiqueta_1, etiqueta_2, ..., etiqueta_n*`}`;
`enum` *nome_enum nome_var*;
`enum` *nome_enum* `{`*etiqueta_1, etiqueta_2, ..., etiqueta_n*`}` *nome_var*;

Tabelas

`<tipo>` *nomeVar*[*dimensao*];
`<tipo>` *nomeVar*[ ] $=\{e_0, e_1, e_2, \cdots, e_n\}$;
`<tipo>` *nomeVar*[*dimensao*] $=\{e_0, e_1, e_2, \cdots, e_n\}$;
em que $n \leq dimensao - 1$ e *dimensao* é um valor constante estritamente positivo.

Definição dinâmica de uma tabela:
`<tipo> *` *nomeVar*;
nomeVar `= new <tipo> [num];`

Para libertar o espaço pedido:
`delete[]` *nomeVar*;

Strings no estilo C:

`char` *nomeVar*[*dimensao*];
`char` *nomeVar*[ ] = `"Texto ao gosto do utilizador";`
`char` *nomeVar*[*dimensao*] = `"Texto ao gosto do utilizador";`
em que o comprimento (útil) do texto deve ser menor que *dimensao*.

Exemplo de leitura duma string no estilo C:
`cin.getline(nomeVar, sizeof(nomeVar));`

Para saber o comprimento duma string no estilo C:
`strlen(nomeVar)`

Strings no estilo C++:

`string` *nomeVar*;
`string` *nomeVar* = "Texto ao gosto do utilizador";

Exemplo de leitura duma string no estilo C++:
`getline(cin, nomeVar);`

Para saber o comprimento duma string no estilo C++:
`nomeVar.length()`

Manipulação de ficheiros – Leitura

Para para aceder a um ficheiro de entrada:
`std::ifstream ficheiro_dados; // Fich. entrada`

Em seguida é preciso associar esta variável com o nome do ficheiro que desejamos ler:
`ficheiro_dados.open("nomefich.ext");` // modo texto por omissão
`ficheiro_dados.open("nomefich.ext", modo);` // modo dado

Ou fazendo tudo de uma só vez:
`std::ifstream ficheiro_dados("nomefich.ext");`
`std::ifstream ficheiro_dados("nomefich.ext", modo);`

Manipulação de ficheiros – Escrita

Para para aceder a um ficheiro para escrita:
`std::ofstream ficheiro_dados;` // Fich. saida

Em seguida é preciso associar esta variável com o nome do ficheiro que desejamos escrever:
`ficheiro_dados.open("nomefich.ext");` // modo texto por omissão
`ficheiro_dados.open("nomefich.ext", modo);` // modo dado

Ou fazendo tudo de uma só vez :
`std::ofstream ficheiro_dados("nomefich.ext");` // modo texto por omissão
`std::ofstream ficheiro_dados("nomefich.ext", modo);` // modo dado
`proteccao` pode ter o valor 0666 (permissão de leitura e escrita para todos).

Bandeira (*flag*)	**Significado**
`std::ios::app`	Acrescenta informação no fim do ficheiro.
`std::ios::ate`	Abre e vai para o fim do ficheiro
`std::ios::in`	Abre o ficheiro para leitura
`std::ios::out`	Abre o ficheiro para escrita
`std::ios::binary`	Abre o ficheiro binário. Se esta bandeira não está activa o ficheiro é considerado como sendo ASCII.
`std::ios::trunc`	Descarta o conteúdo corrente do ficheiro quando aberto para escrita.

Tabela 6: Modo - *open flags*

Algumas funções membro de manipulação:

```
bool ficheiro_dados.close();       // fecha o ficheiro
bool ficheiro_dados.is_open();     // true se o ficheiro está aberto
bool ficheiro_dados.bad();         // true se o bit badbit foi activado
bool ficheiro_dados.fail();        // true se badbit ou failbit foram activados
bool ficheiro_dados.eof() const;   // true se foi activada a flag de fim de ficheiro
char int ficheiro_dados.peek();    // lê e devolve o carácter seguinte sem o extrair
```

Para saltar 1 caracter da entrada:
`char ficheiro_dados.ignore();`

Para saltar n caracteres da entrada ou até encontrar o carácter delim:
`char ficheiro_dados.ignore(n, delim);`

A leitura e escrita FORMATADA consegue-se usando os operadores >> e << , respectivamente.

A leitura NÃO FORMATADA:
fich_entrada.read(data_ptr, tamanho);
Em que:

- *fich_entrada* é um objecto da classe `ifstream` (ou `fstream`) aberto para leitura.
- *data_ptr*: ponteiro (para char) para o local onde deve ser colocada a informação lida.
- *tamanho*: número (int) de octetos a ler.

A escrita NÃO FORMATADA:
fich_saida.write(data_ptr, tamanho);

- *fich_saida* é um objecto da classe `ofstream` (ou `fstream`) aberto para escrita.
- *data_ptr*: ponteiro (para char) para o local onde está a a informação que será colocada no ficheiro
- *tamanho*: número (int) de octetos a escrever.

As funções membro `setf` e `unsetf` podem ser utilizadas para activar ou desactivar flags:
file_var`.setf(flags); // Activa flags`
file_var`.unsetf(flags); // Desactiva flags`
em que *file_var* é do tipo `ofstream` ou `ifstream`,

Bandeiras de conversão	
Flag	**Significado**
std::ios::skipws	Ignora os espaços que precedem a entrada.
std::ios::left	A saída é ajustada à esquerda.
std::ios::right	A saída é ajustada à direita.
std::ios::internal	Preenche uma saída numérica introduzindo um carácter de enchimento entre o sinal ou a base e o próprio número.
std::ios::boolalpha	Apresenta `true` e `false` em vez de 1 ou 0.
std::ios::dec	Apresenta números na base 10, formato decimal
std::ios::oct	Apresenta números na base 8, formato octal.
std::ios::hex	Apresenta números na base 16, formato hexadecimal.
std::ios::showbase	Apresenta o indicador de base no início de cada número.
std::ios::showpoint	Apresenta o ponto decimal em números de vírgula flutuante, quer seja ou não necessário.
std::ios::uppercase	Na conversão para hexadecimal apresenta os dígitos A-F em maiúsculas
std::ios::showpos	Coloca um sinal + antes de todos os números positivos.
std::ios::scientific	Converte todos os números de vírgula
std::ios::fixed	Converte todos o números para a notação de vírgula fixa.

Tabela 7: Bandeiras de conversão

Manipuladores de Entrada/Saída	
Manipulador	**Descrição**
std::setiosflags(long flags)	Escolhe as bandeiras a activar.
std::resetiosflags(long flags)	Desactiva as bandeiras seleccionadas.
std::dec	Apresenta números em formato decimal.
std::hex	Apresenta números em formato hexadecimal.
std::oct	Apresenta números em formato octal.
std::setbase(int base)	Escolhe a base de conversão: 8, 10 ou 16.
std::setw(int width)	Define a largura mínima da saída
std::setprecision(int precision)	Define a precisão de número em vírgula flutuante.
std::setfill(char ch)	Define o carácter de enchimento.
std::ws	Ignora os espaços que precedem a entrada.
std::endl	Apresenta o carácter de fim de linha `'\n'`.
std::ends	Apresenta o carácter de fim de string `'\0'`.
std::flush	Força a saída do que esteja em *buffer*.

Tabela 8: Manipuladores de Entrada/Saída

Mais funções de manipulação de ficheiros:

Seja:
```
ofstream fout("output.txt");
ifstream fin("input.txt");
```

- `tellp()` devolve a posição corrente do cursor de um ficheiro de **saída** (ponteiro para a posição do próximo `put`).
  `cout << fout.tellp();`
- `tellg()` O equivalente a `tellp()` para ficherios de entrada: devolve a posição corrente do cursor de um ficheiro de **entrada** (ponteiro para a posição do próximo `get`).
- `seekp(`*pos*`)` vai para a posição `pos` de um ficheiro de **saída**.
  `fout.seekp(3);` //// Vai para a posição 3
- `seekg(`*pos*`)` vai para a posição `pos` de um ficheiro de **entrada**.
  `fin.seekg(2);` // Vai para a posição 2
- `gcount()` é utilizada com canais de **entrada** devolve o número de caracteres lidos na última operação de entrada.

  É utilizada em conjunção com `get, getline` e `read`.
- `putback()` é utilizada com canais de entrada e coloca de novo na entrada o último carácter lido e coloca o cursor de leitura sobre ele. `cin.putback('g');`

  Está relacionada com a função `peek()`.

Entrada/Saída usando printf() e scanf()

- `int printf ( const char * formato);` envia para o `stdout` (em geral o ecrã) o texto na string apontada por formato (que não contém especificadores de formato), e devolve o número de caracteres enviados.
- `int printf ( const char * formato, ...);` se o `formato` inclui especificadores de formato (sub-sequências começadas por %), os argumentos que se seguem a `format` são inseridos na string substituindo os respectivos especificadores, e a string resultante é enviada para o `stdout`.

A forma completa do especificador de formato:
%[flags][width][.precision][length]specifier
em que *specifier* é um dos elementos na coluna **Especificador** da tabela que se segue.

Lista de *alguns* especificadores de formato de **printf**():

Especificador	**Tipo do argumento**	Saída
c	`char` ou `int` ou `unsigned int`	uma letra
d	`int` ou `short int` ou `char`	número decimal inteiro, com sinal
f	`float`	número decimal real, com sinal
lf	`double`	número decimal real, com sinal
u	`unsigned int` ou `unsigned short int`	número decimal inteiro, sem sinal
lu	`unsigned long int`	número decimal inteiro, sem sinal
s	`char *` (C style string)	cadeia de caracteres
o	`int` ou `unsigned int`	Número inteiro (sem sinal) em octal
x	`int` ou `unsigned int`	Número inteiro (sem sinal) em hexadecimal
e	`float`	Número decimal real no formato *d.ddd*e+*dd*
E	`float`	Número decimal real no formato *d.ddd*E+*dd*
g	opta entre %e e %f, o que conduzir a um número mais pequeno	Número decimal real
G	opta entre %E e %f, o que conduzir a um número mais pequeno	Número decimal real
%	Um % seguido de %	permite escrever o carácter %

flags Justificação à esquerda, colocar ou não o sinal.
width Número mínimo de caracteres a imprimir.
precision Número mínimo de dígitos a imprimir.
length Modifica o comprimento do tipo de dados (se possível).

- `int scanf ( const char * format, ... );` lê dados formatados do `stdin`, de acordo com o parâmetro `format` e armazena a informação na localizações de memória apontados pelos argumentos que se seguem. Esses argumentos adicionais devem apontar para objectos (zona de memória) do tipo especificado pelo formato.

  A função lê e ignora qualquer carácter designado por *Whitespace* (espaço, tabulações e mudanças de linha), o qual funciona como separador de argumentos a ler (com uma excepção).

  A forma completa do especificador de formato (alguns precedidos da *length* "l"):
  %[*][width][length]specifier

Lista de *alguns* especificadores de formato de **scanf**():

Especificador	**Tipo do argumento**	Funcionamento
i	`int` `unsigned int`	Lê qualquer número de dígitos, opcionalmente precedidos de sinal (+ ou -). Por por omissão os dígitos (0-9), o prefixo 0 indica octal (0-7), e o prefixo 0x hexadecimal (0-f)
d, u	`int`, `unsigned int`	Lê qualquer número de dígitos (0-9), opcionalmente precedidos de sinal (+ ou -). d número pode ter sinal, u número sem sinal
o	inteiro na base octal	pode ter sinal
x	inteiro na base hexadecimal	pode ter sinal
f, e, g	número em vírgula flutuante	float: número decimal real, com sinal
lf, le, lg	número em vírgula flutuante	double: número decimal real, com sinal
lu	`unsigned long int`	número decimal inteiro, sem sinal
s	`char *` (C style string)	cadeia de caracteres, pára de ler no primeiro *Whitespace*. Coloca no final o carácter de terminação.
c	Se `width` omisso ou igual a 1	lê um carácter
	Se `width` diferente de 1	lê exactamente `width` caracteres, e guarda-os no local apontado (sem carácter de terminação)

Programa:

```
#include <cstdio>
int main () {
   char apelido [80];
   int i;
   double peso;

   printf ("Apelido: ");
   scanf ("%79s", apelido); // ZZZ não pode ter mais de 79 letras!
   printf ("Idade: ");
   scanf ("%d", &i);
   printf ("Peso: ");
   scanf ("%lf", &peso);
   printf ("%s, pesa %.1lf e tem %d anos.\n", apelido, peso, i); }
```

Execução, com espaços assinalados:

```
Apelido:␣Amaral
Idade:␣45
Peso:␣78.87
Amaral,␣pesa␣78.9␣e␣tem␣45␣anos.
```

Funções de manipulação de Strings no estilo C:

- `char *strcpy (char *dest, const char *orig)`

  Copia a string apontada por *orig* para *dest* (coresponde a `dest <-- orig`). Devolve um ponteiro para a string destino (*dest*).

- `char *strncpy(char *dest, const char *orig, size_t n)`

  Copia para *dest* até *n* caracteres da string apontada por *orig*. Se *orig* tem menos do que *n* caracteres, o espaço restante será preenchido com o octetos a 0. Devolve um ponteiro para a string destino (*dest*).

- `char *strcat(char *dest, const char *orig)`

  Adiciona a string apontada por *orig* no final da string apontada to por *dest* (corresponde a `dest <-- dest + orig`). Devolve um ponteiro para a string destino (*dest*).

- `char *strncat(char *dest, const char *orig, size_t n)`

  Adiciona a *dest* até *n* caracteres da string apontada por *orig* (e no final coloca o carácter de fim de fim de string) Devolve um ponteiro para a string destino (*dest*).

- `int strcmp(const char *str1, const char *str2)`

  Compara a string apontada por *str1* com a atring apontada to *str2* e devolve:
  0 se *str1* igual a *str2* ;
  < 0 se *str1* $<$ *str2*;
  > 0 se *str1* $>$ *str2*

Nota: `size_t` corresponde ao tipo inteiro sem sinal e é o tipo devolvido pela função `sizeof`.

Descrição simplificada de métodos de manipulação de Strings C++:

- `const char* c_str() const;`

  Devolve um ponteiro para uma tabela (interna) que contem a sequência caracteres (devidamente terminada com `\0`) que representa o valor corrente do objecto *string* C++.

  Esta tabela (string C) não pode ser alterada e o seu conteúdo poderá ficar inválido se outros métodos alterarem a string C++ que a mesma representa.

- `int compare (const string& str) const;`

  Compara a *string de comparação* com *str* e devolve:
  0 se a *string de comparação* é igual a *str*;
  < 0 se a *string de comparação* menor que *str* – ou seja se o primeiro carácter diferente for menor na *string de comparação*, ou sendo todos iguais o comprimento da *string de comparação* é o menor.
  > 0 se a *string de comparação* maior que *str* – ou seja se o primeiro carácter diferente for maior na *string de comparação*, ou sendo todos iguais o comprimento da *string de comparação* é o maior.

- `string substr (size_t pos = 0, size_t len = npos) const;`

  Constrói a sub-string que é uma cópia (de parte) do objecto que invoca o método. A sub-string começa no índice *pos* e abrange *len* caracteres seguintes (ou até tingir o fim da string, o que ocorrer primeiro).

  A constante *string::npos* tem o valor do número máximo possível de caracteres numa *string*. Logo, os valores por omissão correspondem a criar uma cópia do objecto que invoca o método (de 0 até ao fim).